Improved Algorithm for Seamlessly Creating Infinite Loops from a Video Clip, while Preserving Variety in Textures

Kunjal Panchal

University of Massachusetts, Amherst
kpanchal@umass.edu

December 12, 2019

Abstract

This project implements the paper "Video Textures" by Szeliski [1]. The aim is to create a "Moving Picture" or as we popularly call it, a GIF; which is "somewhere between a photograph and a video". The idea is to input a video which has some repeated motion (the texture), such as a flag waving, rain, or a candle flame. The output is a new video that infinitely extends the original video in a seamless way. In practice, the output isn't really infinte, but is instead looped using a video player and is sufficiently long as to appear to never repeat.

Our goal from this implementation was to: improve distance metric by switching from a crude sum of squared distance to most sophisticated wavelet-based distance; add intensity normalization, cross-fading and morphing to the suggested basic algorithm. We also experiment on the trade-off between variety and smoothness.

I. Introduction

In current age of technology, GIFs[Graphics Interchange Format] are popular because of their relatively smaller size compared to other formats. Moreover, loading images online is quicker without losing its quality. And they simply convey the message better than a static image. In the paper "Video Textures" by Schodl, Szeliski, and Essa [1]; they created a ''new type of medium" called a video texture, which is a continuous infinitely varying stream of images, same concept as a GIF. They presented techniques for analyzing a video clip to extract its structure, and for synthesizing a new, similar looking video of arbitrary length, which is seamless. The goal is that the method needs only few frames out of the whole clip, it works infinitely continuously with varying patterns to give an illusion of arbitrarily long video clip. We implement this method from scratch.

But some clips like water, fire, grass in the wind, candle flames, flags face some jitter (conspicuous jumps between frames) - paper mentions this can solved by cross-fading, but they have not implemented it, we will, in this project, try to remove all the jitters; this is easily fixed by modifying Distance Matrix D so that the pairwise distance between frames also considers a few frames around it. This can be achieved by filtering D using a length 2 or 4 filter with weights set to the binomial coefficients (to approximate a Gaussian distribution, with the correct width). Next, in video clips featuring time lapses, the lighting changes too drastically when we choose only few frames from many. This can be solved by some pre-processing – normalizing intensities.

Then, there are some future work suggestions in the end: use a better distance metric, better blending and maintaining variety. We implement Chebyshev Distance and Wavelet-based Distance; compare the results. Then, we use morphing for better blending, that can do away with jitters. For maintaining variety, we define a parameter which controls randomness. It penalizes a lack of variety in the generated sequences.

Such a parameter would enforce that most (if not all) of the frames of the given input sequence are sometimes played and probabilistically vary the generated order of frames.

II. Related Background Work

Our main reference for the base algorithm is the original paper on Video Textures [1] where they called a GIF, a video texture depicting a certain repeatable pattern and their extensions which include the display of dynamic scenes on webpages, the creation of dynamic backdrops for special effects and games, and the interactive control of video-based animation. One of our main enhancements are to create time-lapses using the same base algorithm, we looked for ways to normalize the color intensity of a video as a pre-processing step. The paper on "Efficient fluorescence image normalization for time lapse movies" [2] provides great insight where the

concerned lighting is fluorescent, which infers a time-dependent back-ground signal and the image gain without the use of additional fluorescent substances. They first tiled the full image into small sub-images and determined background tiles by clustering the statistical moments of the individual intensity distributions. For each image, they interpolated the full background from the identified tiles and thus reconstituted the time-dependent background image. Second, they estimated the time-independent image gain from the background tiles of all pixels and all time points.

A thesis on "Probabilistic Time Lapse Video" [3] provides basic to advanced histogram techniques for color normalization in images. The thesis also talked about reduction of noise, jitter removal and normalization of color levels methods being employed in the preprocessing stage to clean the images and provide a suitable starting point for the implementation of the time lapse video creation algorithms.

Later, we explored techniques of distance measurements other than sum of squared distance. We first used wavelet transforms to compress each frame into its second level representation. The paper on "A Study on Wavelet Compression Images Based on Global Threshold" [4] analyzes the combination between different wavelet filters to select one that give the best compression ratio. In that work they proposed a new type of global threshold to improve the wavelet compression technique. The aim was to maintain the retained energy and to increase the compression ratio. The work on "Real-time compression and decompression of wavelet-compressed images" [5] talks about compressing images stored as collections of tiled line textures representing breadth-first trees and then the image is decompressed directly on a GPU employing a microcode pixel shader.

To cluster the compressed images, there is lot of literature available on online k-means algorithms [6, 7, 8, 9].

III. Approach and Algorithm Analysis

There are two main parts to the algorithm. First, the frames, which are unique enough to create a GIF that will include most of the variety found in the video clip, must be extracted from the input clip. Second, a new clip/ GIF that synthesizes the separate frames must be created.

But, the essential concept behind the algorithm is this: given a frame of a clip, we can select a plausible next frame by picking a similar frame, which might be similar enough to be in continuity of the previous frame, but not so similar that we don't get motion or variety in the GIF at all; to the one that would have been played in the original video clip. This next frame may not be the actual next frame in the input, but it may be. In this way, we can infinitely and smoothly extend a video.

i. Extracting the Video Patterns

To extract the video patterns, we need to compute how much alike the pairs of frames are to each other. This can be achieved by calculating the sum of squared difference between each pair of frames, and storing the results in a distance matrix, D. (More sophisticated ways are discussed in the upcoming sections).

From there, we can compute a probability matrix, P, which assigns probabilities between pairs of frames. P can then be used to calculate the next frame in the output video given a current frame: since a frame is a row, we discretely sample the next frame using the probability distribution across a row of P. The pseudo-code is below:

Algorithm 1 Extracting the Next Frame

```
1:  procedure ExtractFrames
2:      # Construct D, the distance matrix
3:      D ← pairwise distance between all frames,
4:          usingSSD
5:      shift D to the right by 1,
6:      to align the next frame with potential new
7:      frames
8:      # Construct D, the distance matrix
9:      σ ← average of non-zero D values
10:         ∗ SIGMA_MULTIPLE.
11:     P ← exp(−D/σ).
12:     Normalize P so that the sum of row is 1
```

We discuss how to set the SIGMA_MULTIPLE correctly in the next sections.

ii. Preserving Dynamics

In some cases, the input video has a fluid motion that the GIF should preserve. The paper [1] gives the example of a pendulum swinging in Figure 1: the algorithm described in 1 doesn't account for the fact that original video has a side-to-side motion. So, the resulting texture may jitter back and forth since there's no distinction that the next frame may

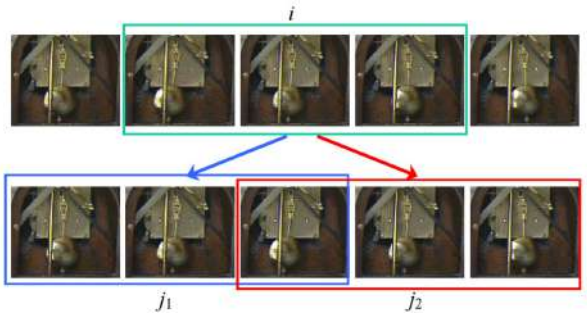


Figure 1: *Effect and Need of Dynamics Preservation. A GIF of a pendulum should preserve the direction in which it is moving*

have come from the left side or the right side in the original video.

This is easily fixed by modifying D so that the pairwise distance between frames also considers a few frames around it. This can be achieved by filtering D using a length 2 or 4 filter with weights set to the binomial coefficients (to approximate a Gaussian distribution, with the correct width). The pseudo-code for this modification is below:

Algorithm 2 Weighted Probability Associated with Each Frame

```
procedure WeightFrames
    # Modify D, to preserve motion
    w ← a 2 or 4 length filter
        with binomial weights
    w ← diag(w)
    filter D with w
    crop D along the edges due to the filter
    # From here, continue on with making P as
    described above
```

The impact on D can be visualized below in Figure 2:

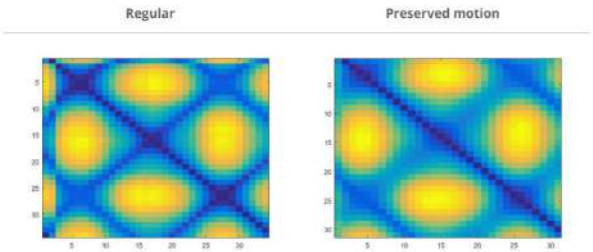


Figure 2: *Motion Preservation: Blurring can be observed along the diagonals, and D is slightly smaller in overall size due to the cropping*

IV. Writing the Video Loops

Once suitable transitions have been identified, we can then generate the GIF. The original paper [1] describes two approaches to do so: random playback, and video loops. Random playback was just sticking any two frames which were close enough distance-wise, and it didn't provide much logic or sophistication for our main goal of jitter removal. So here, we just discuss a more logical method, which gave us satisfying results.

The motivation behind this method is to preselect a sequence of loops that can be repeated, so that the resulting clip smoothly repeats when played on a conventional video player's repeat setting. We attempted to implement dynamic programming algorithm and scheduling algorithm. We had to prune the input video clip, because in many cases, the very start and the end of the video clip are very different from the actual content of the clip.

In dynamic algorithm, we have a collection of potential frames; we know that the first and the last frame must be the one and the same. So we take a potential first frame, and go backward towards to same frame by using the concepts from "Longest Common Subsequence", where a subsequence is a sequence that can be derived from another sequence by deleting some elements without changing the order of the remaining elements. Longest common subsequence (LCS) of 2 sequences is a subsequence, with maximal length, which is common to both the sequences.

Here, we just change the "characters" with "frames" and find a longest sequence. The reason we want it to be longest is because we need to have as many unique frames in the GIF as possible, to preserve the variety of the original clip.

V. Methods

Till now we talked about the basic algorithm with few improvements here and there, here I will focus on mainly the 5 significant improvements I have implemented to the original methodology. Those 5 ideas are as follows:

1. Smoothening the Jitters: Cross-Fading
2. Smoothening the Jitters: Blending through Morphing
3. For drastic changes in light intensities: Normalizing

4. Better Distance Metric: Wavelet-transform based Clustering

5. Variety Preservation: Finding the correct hyper-parameters

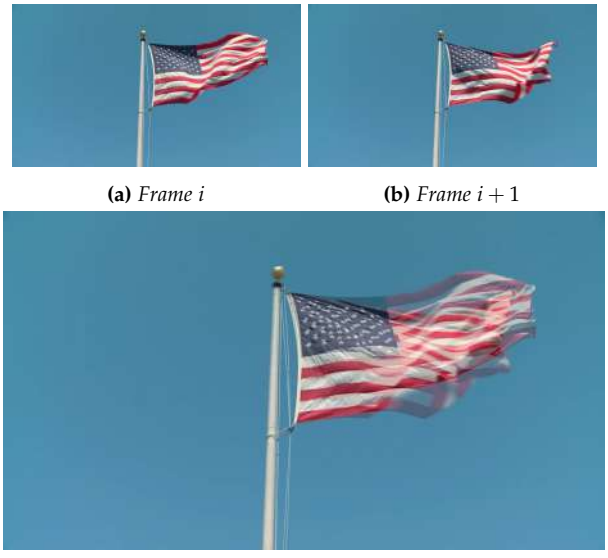
i. Smoothening the Jitters with Cross-Fading

Many times, we will find that there doesn't exist a smooth transition between two frames. Which will result in conspicuous jitters. To avoid that, we might be lax in our choice of potential next frames, but that might make our loop/ GIF too short or with much lesser number of unique frames, which won't capture the essence of the clip.

One solution is to prefer variety over smoothness, and remove jitters by cross-fading each transition between frames.

The trick is to not show frame i for t milliseconds and then switch to frame $i+1$ while completely "turning off" the frame i; but we have to gradually decrease the opacity (alpha channel) of frame i, while we have started showing frame $i+1$, that too, by gradually *increasing* the opacity or the alpha value.

That way that jitter between the two frames will not be as apparent as it previously was. Figure 3 shows the results of cross-fading between two frames in a clip where a flag in waving in the wind.



(a) *Frame i* **(b)** *Frame* $i+1$

(c) *Transition between Frame i and* $i+1$

Figure 3: *Example of Cross-Fading where we decrease alpha value of Frame i as we increase alpha value of* $i+1$

ii. Smoothening the Jitters with Morphing

A more advanced technique would be to implement morphing, as it would actually transform one frame to the next one, instead of just making the transitional jitters less conspicuous.

We can warp Frame i features to Frame $i+1$ features by homography transforms. We first extracted SIFT features from Frames i and $i+1$.

The SIFT features are extracted as shown in Figure 4.

Figure 4: *Feature Extraction of source and destination frame, we can warp the geometry of those features in a way that results in smoother transition by inserting the intermediate transition frames*

The algorithm for interpolating intermediate frames as given in

Algorithm 3 Interpolation of Intermediate Frames

```
procedure IntermediateFrames
for: r = each row of Frame i
    for: c = each column of Frame i+1
    q ← Frame1[r, c]
    p ← Frame2[r, c]
    # The intermediate FrameN is a combination
    # of the Frame1 and Frame2
    FrameN[r, c] ← p + (1 − i/n) ∗ (q − p)
    end
end
# Repeat this N times
# If N is larger, smoother transition
  but larger sized GIF
# If N is smaller, less smoother transition
  but smaller sized GIF
```

iii. Normalizing the Light Intensity

To create a GIf of timelapses of a city, the GIF will start with sunny sky and end with night sky, and must be linear so there aren't drastic line changes in the flow.

To avoid the sudden changes with respect to sky color, we can first normalize the colors of whole video clip and then apply the same algorithm to extract and synthesize the frames.

"Efficient fluorescence image normalization for time lapse movies" [2] talks about the same topic but focuses on fluorescent colors. A theses from University of Edinburgh, "Probabilistic Time Lapse Video" [3] expands the idea of Histogram based contrast stretching and Histogram Equalization and apply a color transformation as follows:

$$\text{Frame}'_{i(r,c)} = \begin{cases} 0, & \text{Frame}_{i(r,c)} \leq V_{min} \\ 255 \cdot \frac{\text{Frame}_{i(r,c)} - V_{min}}{V_{max} - V_{min}}, & V_{min} < \text{Frame}_{i(r,c)} < V_{max} \\ 255, & \text{Frame}_{i(r,c)} \geq V_{min} \end{cases}$$

where $Frame_{i(r,c)}$ and $Frame'_{i(r,c)}$ are frames before and after the intensity stretching respectively; V_{min} and V_{max} are the color values equivalent to the upper and lower bounds of the distribution 1% and 99% [3].

iv. Distance Measurement with Wavelet based Clustering

I tried to explore better distance metric than sum of squared distance and focused on binning very similar images together.

Using a wavelet transform, the wavelet compression methods are adequate for representing transients, such as high-frequency components in two-dimensional images. This means that the transient elements of a data signal can be represented by a smaller amount of information than would be the case if some other transform, such as the more widespread discrete cosine transform, had been used.

So idea here to just get the high level details of each frame and cluster all the similar images in one cluster, while increasing the interclass distance and decreasing the intraclass distance. After using wavelet transforms, we use K-means algorithm. It can be interpreted as a greedy algorithm for approximately minimizing a loss function related to data compression.

Figure 5 shows a two level compression of a single frame from our flag clip.

After compressing all the frames, we can run K-means algorithm. The choice of K is arbitrary but keeping the computational resources in mind and size of the output GIF in check, I have decided to keep K between 1 to 10. I ran K-means for $K = 1$ to 10, and visualized and quantified the clusters to pick the optimal K manually which minimizes the intraclass distance and maximizes the interclass distance.

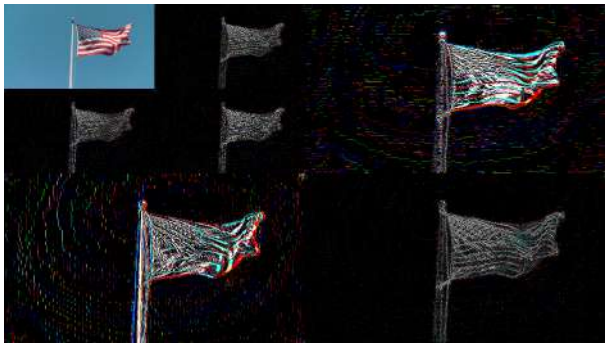


Figure 5: *A two-level 2D discrete wavelet transform*

Now that the clusters are decided, we need to choose a strategy to pick frames which will play in continuity.

One strategy is to pick one initial frame, then pick the next one from the nearest cluster which is not the same cluster as the current frame came from. And the next to next frame from the cluster which was not previous or previous to previous. That way we will get variety and smooth transition as the next frame will be nearer to the current but not so near that it won't induce the variety.

Paper "A Study on Wavelet Compression Images Based on Global Threshold" [4] gives a good insight into this.

v. Hyper-parameters

The probability matrix P is created using an exponential function and dividing by a constant, σ. The paper [1] notes that σ is (often, but in our case always) set to a small multiple of the average non-zero D values. The user-set parameter SIGMA_MULTIPLE controls σ further: smaller values of σ force the best transitions to be taken, and larger values allow for more randomness. We typically set SIGMA_MULTIPLE to 0.05.

VI. Results

First we show the results of using SSD distance metric. Figure 6 show raw distance between Frame i and each Frame j where $j \in \{1, \ldots, \#\text{NumberOfFrames}\}$

We must remember that we need to make a seamless loop out of the video clip, so we shift the distance matrix circularly to get the rows with minimum difference between the first cell and the last cell. Figure 7 shows the circular shift on the distance matrix.

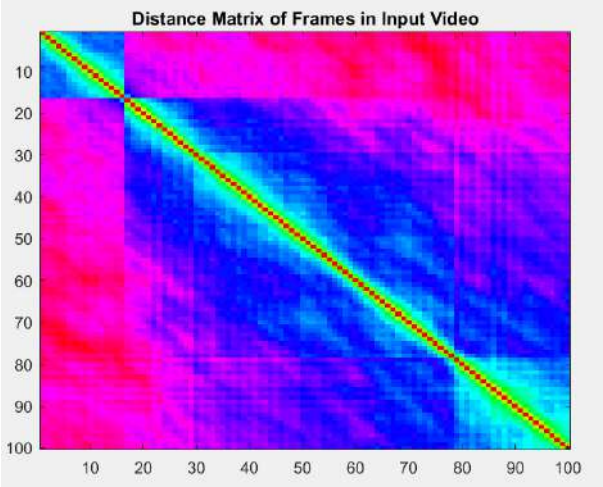


Figure 6: *Distance Matrix of Frames in Input Video*

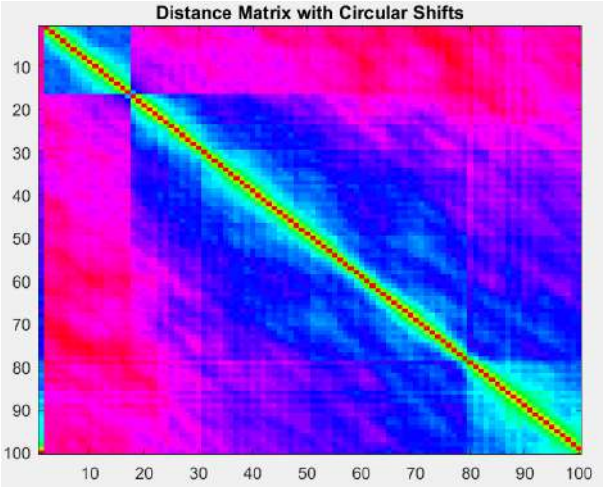


Figure 7: *Distance Matrix with Circular Shifts*

Now we calculate the weights of neighbor frames so we can assign probability of potential next frame to preserve continuous motion. Figure 8 shows four weighted neighbors in one dimension and Figure 9 shows the same in two dimensions.

We use $P = \exp(-D/\sigma)$ to get the weighted distance matrix as shown in Figure 10.

Besides SSD, we also used wavelet based K-means clustering, the 5 clusters for the flag clip are shown in Figure 11 along with the uncompressed version of few frames representing that cluster.

Figure 12 shows few frames in sequence of the generated loop.

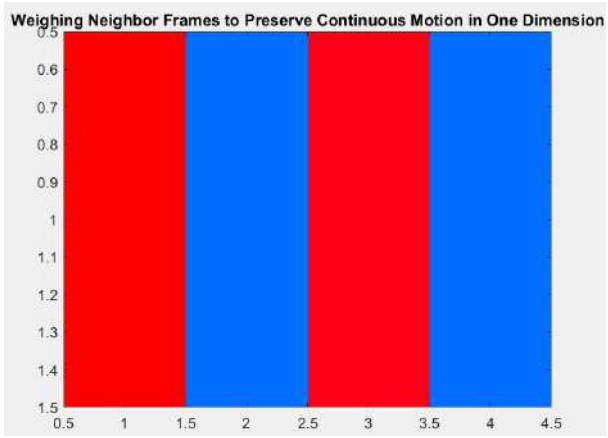


Figure 8: *Weighted Neighbor Frames to Preserve Continuous Motion in One Direction*

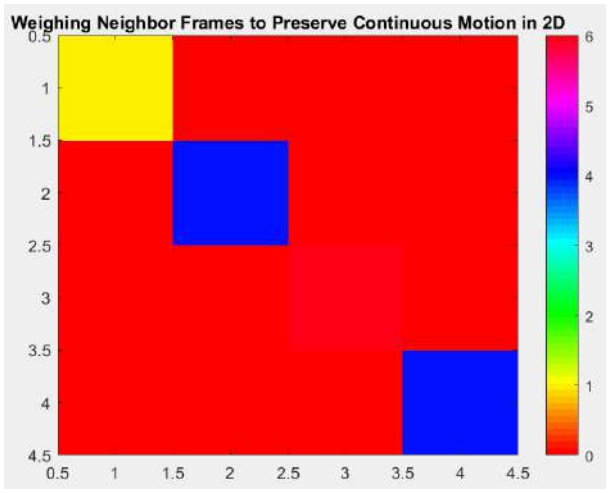


Figure 9: *Weighted Neighbor Frames to Preserve Continuous Motion in Two Direction*

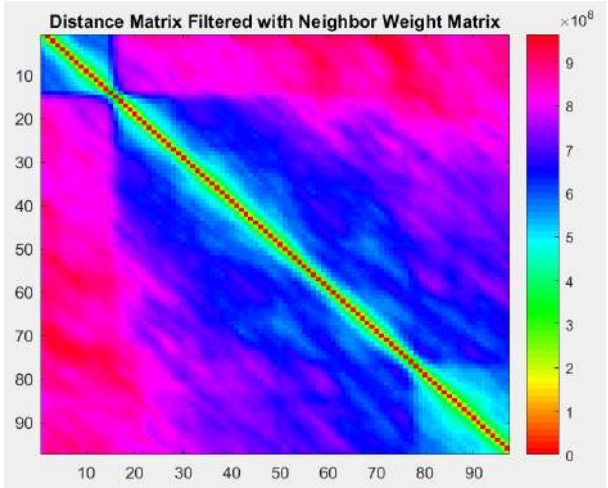


Figure 10: *Distance Matrix with Neighbor Weight Matrix*

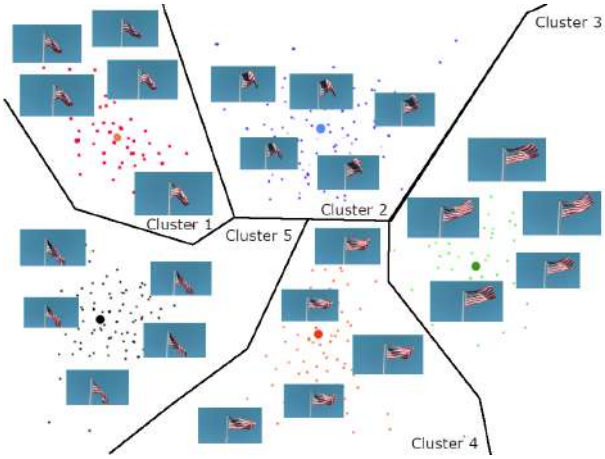


Figure 11: *Example of 5-means clustering with Original Frames*

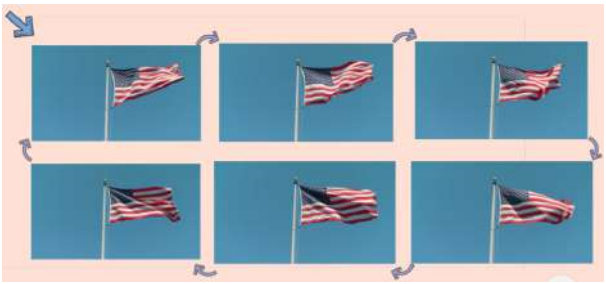


Figure 12: *Example Transition of few Frames from the Generated GIF*

VII. Discussion

i. Other Video Clips

Snowfall makes for a great video texture as it's very hard to perceive where the loop happens. Snowflakes all pretty much look the same, so the frame transitions are very smooth. Furthermore, the snow is falling quickly which makes it harder to notice any poorer transitions that may be made. See Figure 13

Figure 13: *A still from snowfall GIF*

The waterfall performs much better with our cross-fading addition, than it did with the original base algorithm. Cross-fading works since water doesn't have a sharp edges that would cause the fade stand out. See Figure 14

Figure 14: *A still from waterfall GIF*

For timelapses, our normalization of light intensity method actually produced smooth results, we can create Cinemagraphs out of this method [**?**], which are high quality GIFs that are smoothly looped, but they were manually made till now and didn't require well-defined patterns.

Figure 15: *A still from city timelapse GIF*

ii. Future Work

Our algorithm only takes into account local neighbors and they are only tried and tested on relatively short video clips ($\tilde{2}$ minutes). For a longer clip, global neighborhood method might require lot of computational resources.

Also, many clips might not have a repeatable pattern at all. And thus, our algorithm might not get enough material to create a smooth seamless GIF, resulting into jitters and jumps and looping around a very small portion of the clip.

References

[1] A. Schödl, R. Szeliski, D. Salesin, and I. Essa, "Video textures," *Proc. of ACM SIGGRAPH*, vol. 2000, 07 2000.

[2] M. Schwarzfischer, C. Marr, J. Krumsiek, P. Hoppe, T. Schroeder, and F. Theis, "Efficient fluorescence image normalization for time lapse movies," 01 2011.

[3] J. L. R. Ortiz, "Probabilistic time lapse video," p. 100, 01 2008.

[4] C. Pl, "A study on wavelet compression images based on global threshold," *International Journal*

of Applied Engineering Research, vol. 10, pp. 408–411, 04 2019.

[5] M. Boulton, "Real-time compression and decompression of wavelet-compressed images," 05 2013.

[6] V. Cohen-Addad, B. Guedj, V. Kanade, and G. Rom, "Online k-means clustering," 09 2019.

[7] E. Liberty, R. Sriharsha, and M. Sviridenko, "An algorithm for online k-means clustering," 12 2014.

[8] P. Chen, B. Jin, X. Zhu, and M. Fang, "Online k-means algorithm for background subtraction," 01 2015.

[9] P. Sharma and A. Sharma, "Online k-means clustering with adaptive dual cost functions," pp. 793–799, 07 2017.